Design Patterns

MSc in Computer Science

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Swift in Context

Java Example

- Java algorithm to filter a list of strings
- Only printing those shorter than 3 (in this test case).

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

Groovy 1

- Also a valid Groovy program...

```
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

Groovy 1

- Do we need generics?
- What about semicolons...
- Should standard libraries be imported?

```
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

Groovy 2

```
class Erase
{
  public static void main(String[] args)
  {
    List names = new ArrayList()
    names.add("Ted")
    names.add("Fred")
    names.add("Jed")
    names.add("Ned")
    System.out.println(names)
    Erase e = new Erase()
    List short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (String s : short_names)
    {
      System.out.println(s)
    }
  }

  public List filterLongerThan(Liststrings, length)
  {
    List result = new ArrayList();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

Groovy 2

- Do we need the static types?
- Must we always have a main method and class definition?
- Consistency (size or length)?

```
class Erase
{
  public static void main(String[] args)
  {
    List names = new ArrayList()
    names.add("Ted")
    names.add("Fred")
    names.add("Jed")
    names.add("Ned")
    System.out.println(names)
    Erase e = new Erase()
    List short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (String s : short_names)
    {
      System.out.println(s)
    }
  }

  public List filterLongerThan(Liststrings, length)
  {
    List result = new ArrayList();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

Groovy 3

```
def filterLongerThan(strings, length)
{
  List result = new ArrayList();
  for (String s : strings)
  {
    if (s.length() < length + 1)
    {
      result.add(s)
    }
  }
  return result
}

List names = new ArrayList()
names.add("Ted")
names.add("Fred")
names.add("Jed")
names.add("Ned")
System.out.println(names)
List short_names = filterLongerThan(names, 3)
System.out.println(short_names.size())
for (String s : short_names)
{
  System.out.println(s)
}
```

Groovy 3

- Should we have a special notation for lists?
- And special facilities for list processing?

```
def filterLongerThan(strings, length)
{
  List result = new ArrayList();
  for (String s : strings)
  {
    if (s.length() < length + 1)
    {
      result.add(s)
    }
  }
  return result
}

List names = new ArrayList()
names.add("Ted")
names.add("Fred")
names.add("Jed")
names.add("Ned")
System.out.println(names)
List short_names = filterLongerThan(names, 3)
System.out.println(short_names.size())
for (String s : short_names)
{
  System.out.println(s)
}
```

Groovy 4

```
def filterLongerThan(strings, length)
{
  return strings.findAll {it.size() <= length}
}

names = ["Ted", "Fred", "Jed", "Ned"]
System.out.println(names)
List short_names = filterLongerThan(names, 3)
System.out.println(short_names.size())
short_names.each {System.out.println(it)}
```

Groovy 4

- Method needed any longer?
- Is there an easier way to use common methods (e.g. println)?
- Are brackets always needed?

```
def filterLongerThan(strings, length)
{
  return strings.findAll {it.size() <= length}
}

names = ["Ted", "Fred", "Jed", "Ned"]
System.out.println(names)
List short_names = filterLongerThan(names, 3)
System.out.println(short_names.size())
short_names.each {System.out.println(it)}
```

Groovy 5

```
names = ["Ted", "Fred", "Jed", "Ned"]
println names
short_names = names.findAll{it.size() <= 3}
println short_names.size()
short_names.each {println it}
```

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

```groovy
names = ["Ted", "Fred", "Jed", "Ned"]
println names
short_names = names.findAll{it.size() <= 3}
println short_names.size()
short_names.each {println it}
```

Java vs Groovy?

Java Example -> **XTend**

- Unlike Groovy - this is NOT an XTend Program

```
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

def & var

```
import java.util.ArrayList;
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    var Erase e = new Erase();
    var List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  def List<String> filterLongerThan(List<String> strings, int length)
  {
    var List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

Are semicolons necessary?

No semicolons

```
import java.util.ArrayList;
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var List<String> names = new ArrayList<String>()
    names.add("Ted")
    names.add("Fred")
    names.add("Jed")
    names.add("Ned")
    System.out.println(names)
    var Erase e = new Erase()
    var List<String> short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (String s : short_names)
    {
      System.out.println(s)
    }
  }

  def List<String> filterLongerThan(List<String> strings, int length)
  {
    var List<String> result = new ArrayList<String>()
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

Can some types be inferred?

Type inference

```
import java.util.ArrayList;
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var names = new ArrayList<String>()
    names.add("Ted")
    names.add("Fred")
    names.add("Jed")
    names.add("Ned")
    System.out.println(names)
    var e = new Erase()
    var  short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (s : short_names)
    {
      System.out.println(s)
    }
  }

  def filterLongerThan(List<String> strings, int length)
  {
    var result = new ArrayList<String>()
    for (s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

What about Collection Literals?

Collection Literals

```
import java.util.ArrayList;
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    System.out.println(names)
    var e = new Erase()
    var  short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (s : short_names)
    {
      System.out.println(s)
    }
  }

  def filterLongerThan(List<String> strings, int length)
  {
    var result = new ArrayList<String>()
    for (s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

Can Lambas simplify code?

Lambdas

What are List Comprehensions?

```
import java.util.ArrayList;
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    System.out.println(names)
    var e = new Erase()
    var  short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    short_names.forEach[System.out.println(it)]
  }

  def filterLongerThan(List<String> strings, int length)
  {
    val result = new ArrayList<String>()
    strings.forEach[ if (it.length() < length + 1)
      {
        result.add(it)
      }]
    result
  }
}
```

Filters/List Comprehensions

```
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    System.out.println(names)
    var e = new Erase()
    var  short_names = e.filterLongerThan(names, 3)

    System.out.println(short_names.size())
    short_names.forEach[System.out.println(it)]
  }

  def filterLongerThan(List<String> strings, int length)
  {
    val list = strings.filter[it.length() <= 3]
    list
  }
}
```

Do we need the class Erase at all?

Final Version

```
class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    println(names)
    var  short_names =  names.filter[it.length() <= 3]
    println(short_names.size())
    short_names.forEach[println(it)]
  }
}
```

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

java

```
class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    println(names)
    var  short_names =  names.filter[it.length() <= 3]
    println(short_names.size())
    short_names.forEach[println(it)]
  }
}
```

xtend

```
names = ["Ted", "Fred", "Jed", "Ned"]
println names
short_names = names.findAll{it.size() <= 3}
println short_names.size()
short_names.each {println it}
```

groovy

Java Example

- Java algorithm to filter a list of strings
- Only printing those shorter than 3 (in this test case).

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

Swift

```
import Foundation

class Erase
{
  func main()
  {
    var names:String[] = String[]()
    names.append ("ted")
    names.append ("fred")
    names.append ("jed")
    names.append ("ned")
    println(names)
    var short_names:String[] = filterLongerThan(names, length:3)
    for name:String in short_names
    {
      println (name)
    }
  }

  func filterLongerThan (strings : String[], length : Int) -> String[]
  {
    var result:String[] = String[]()
    for s:String in strings
    {
      if countElements(s) < length + 1
      {
        result.append(s)
      }
    }
    return result
  }
}

var erase:Erase = Erase()
erase.main()
```

Swift

- Type Inference

```
import Foundation

class Erase
{
  func main()
  {
    var names = String[]()
    names.append ("ted")
    names.append ("fred")
    names.append ("jed")
    names.append ("ned")
    println(names)
    var short_names = filterLongerThan(names, length:3)
    for name in short_names
    {
      println (name)
    }
  }

  func filterLongerThan (strings : String[], length : Int) -> String[]
  {
    var result = String[]()
    for s in strings
    {
      if countElements(s) < length + 1
      {
        result.append(s)
      }
    }
    return result
  }
}

var erase = Erase()
erase.main()
```

Swift

- Literals

```
import Foundation

class Erase
{
  func main()
  {
    var names = ["ted", "fred", "jed", "ned"]
    var short_names = filterLongerThan(names, length:3)
    for name in short_names
    {
      println (name)
    }
  }

  func filterLongerThan (strings : String[], length : Int) -> String[]
  {
    var result = String[]()
    for s in strings
    {
      if countElements(s) < length + 1
      {
        result.append(s)
      }
    }
    return result
  }
}

var erase = Erase()
erase.main()
```

Swift

- Closures

```
import Foundation

class Erase
{
  func main()
  {
    var names = ["ted", "fred", "jed", "ned"]
    var short_names = names.filter { countElements($0) < 4 }
    for name in short_names
    {
      println (name)
    }
  }
}

var erase = Erase()
erase.main()
```

Swift

- Final version

```
import Foundation

var names = ["ted", "fred", "jed", "ned"]
println(names)
var short_names = names.filter { countElements($0) < 4 }
println(short_names)
```

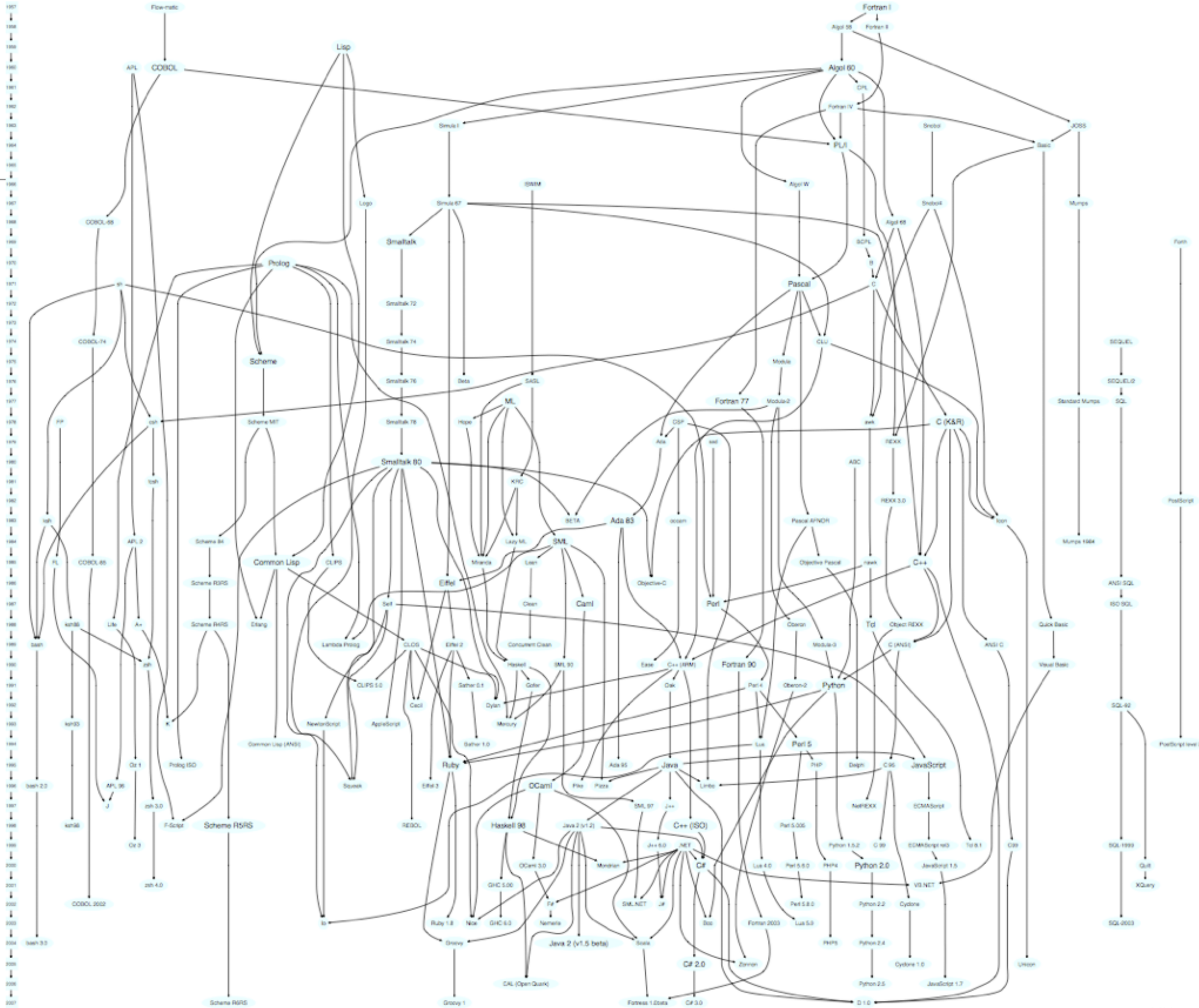
```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

```groovy
names = ["Ted", "Fred", "Jed", "Ned"]
println names
short_names = names.findAll{it.size() <= 3}
short_names.each {println it}
```

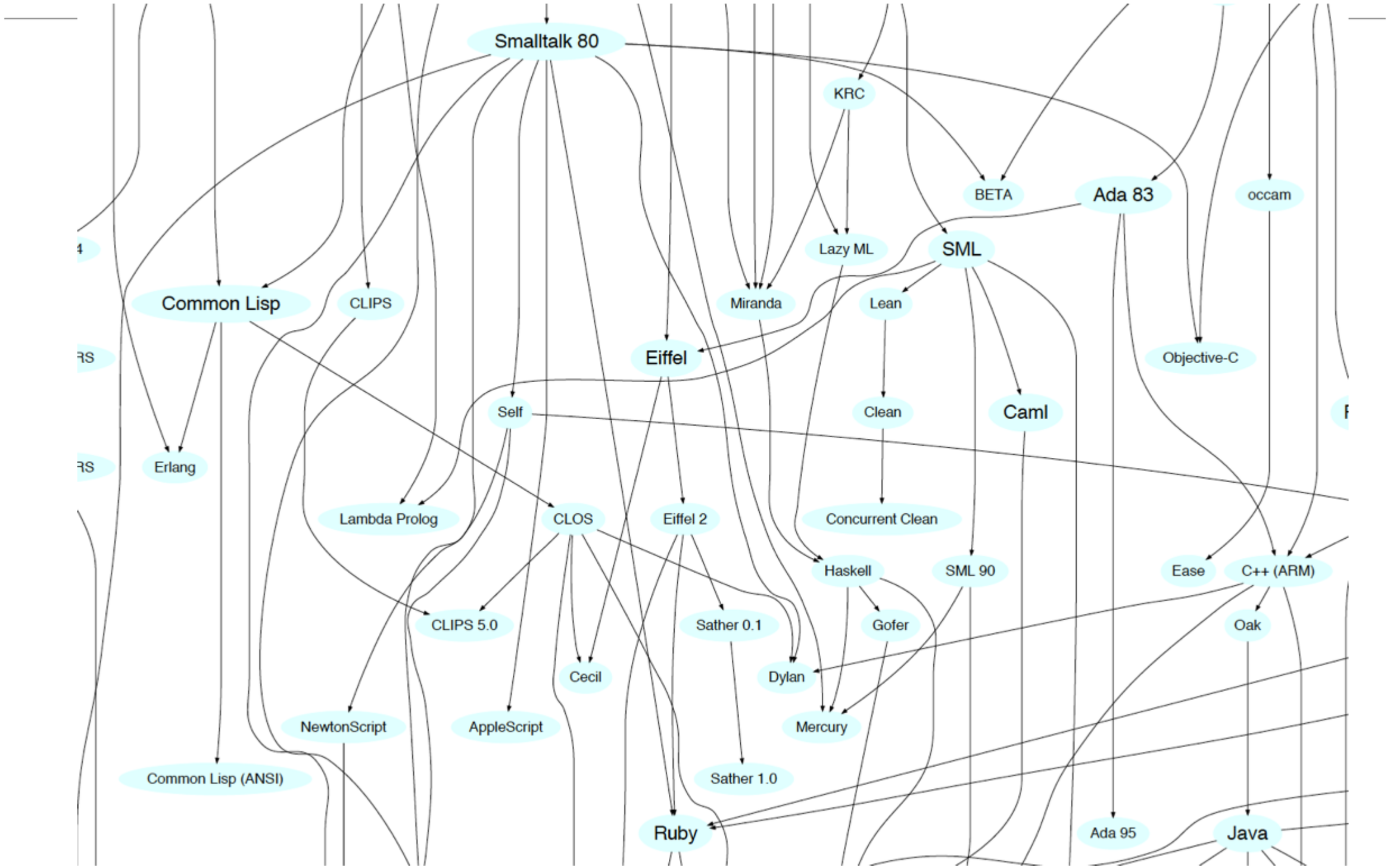
```groovy
var names = #["Ted", "Fred", "Jed", "Ned"]
println(names)
var short_names =  names.filter[it.length() <= 3]
short_names.forEach[println(it)]
```

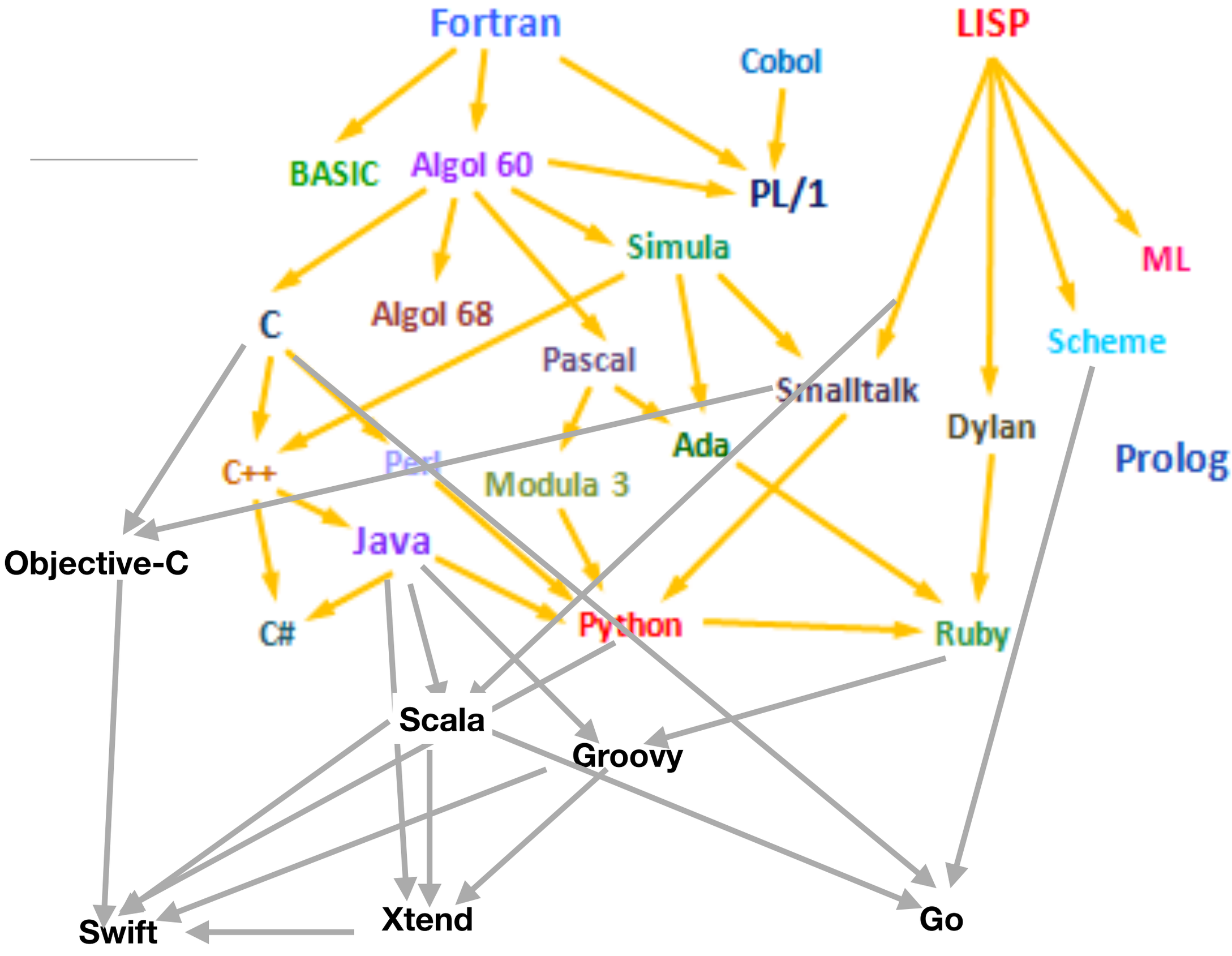
```groovy
var names = ["ted", "fred", "jed", "ned"]
println(names)
var short_names = names.filter { countElements($0) < 4 }
println(short_names)
```

Family Tree (3)

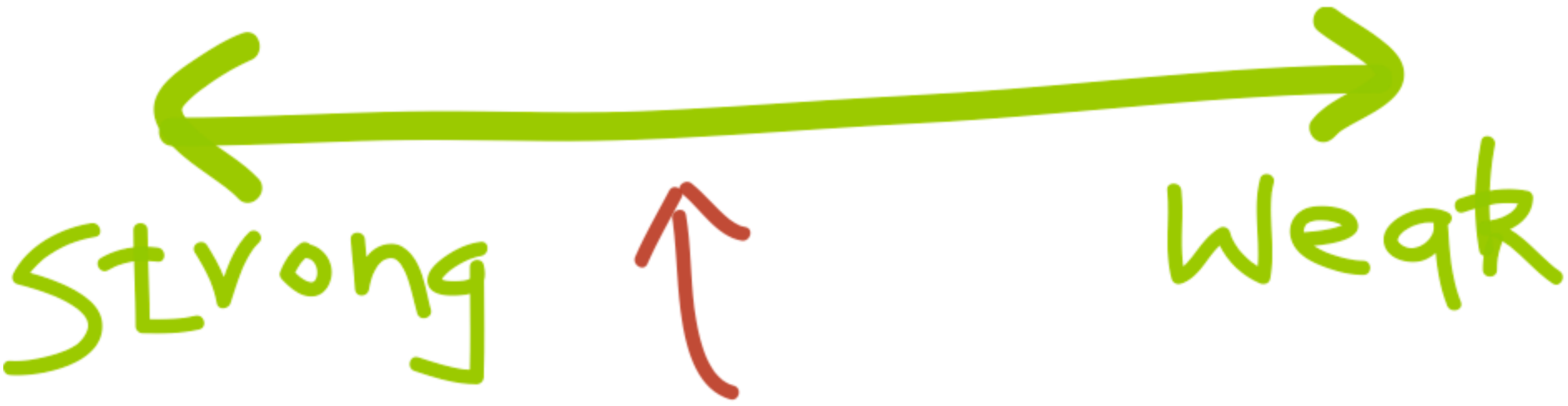


Smalltalk Cluster




Fortran
LISP
Cobol
BASIC
Algol 60
PL/1
Simula
ML
C
Algol 68
Scheme
Pascal
Smalltalk
Dylan
Ada
Prolog
C++
Perl
Modula 3
Java
Objective-C
C#
Python
Ruby
Scala
Groovy
Swift
Xtend
Go

Amount of type checking enforced by the compiler vs. leaving it to the runtime



http://blogs.agilefaqs.com/2011/07/11/dynamic-typing-is-not-weak-typing/

Another Approach to Types?

- *Type Inference* : the compiler draws conclusions about the types of variables based on how programmers use those variables.
  - Yields programs that have some of the conciseness of Dynamically Typed Languages
  - But - decision made at *compile time*, not at *run time*
  - More information for static analysis - refactoring tools, complexity analysis. bug checking etc...
- Haskell, Scala, **Xtend**

```
object InferenceTest1 extends Application
{
  val x = 1 + 2 * 3          // the type of x is Int
  val y = x.toString()       // the type of y is String
  def succ(x: Int) = x + 1   // method succ returns Int values
}
```

‘Pragmatic’ Languages

- Python
- Ruby
- Smalltalk
- Groovy
- Javascript
- PHP
- Scala
- Go
- Swift
- Java
- C#
- C
- C++
- Objective-C

Typing Spectrum